Pipelining the Fast Multipole Method over a Runtime System

Emmanuel Agullo, Bérenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, Toru Takahashi

Pipelining the Fast Multipole Method over a Runtime System

Emmanuel Agullo[a], **Berenger Bramas** [a], Olivier Coulaud[a], Eric Darve[b], Matthias Messner[a], Toru Takahashi[c]

CSE 2013 - February 26

[a] INRIA Bordeaux – Sud-Ouest, France
[b] Institute for Computational and Mathematical Engineering, Stanford University
[c] Department of Mechanical Science and Engineering, Nagoya University

Outline

- Overview of the FMM
- FMM on a runtime system
- Results and performance analysis
- Conclusion and Perspectives

Chebyshev Fast Multipole Method Overview

N-body problems

Example, interactions between planets in a galaxy:

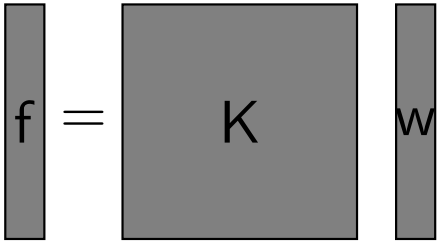
$N(N-1)$ interactions $\rightarrow$ **Complexity is** $0(N^2)$**.**

- Potentials (Kernel $\approx 1/r$)
  - Electrostatic $\longrightarrow$ Molecular dynamics
  - Gravitational $\longrightarrow$ Astrophysics
  - but also electromagnetism, fluid dynamics, VLSI capacitance, ...

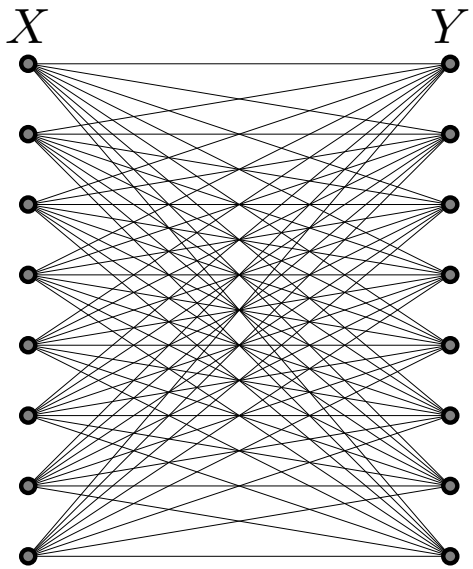
FMM : Motivations

- Pairwise interactions among N bodies.

$$f_i = \sum_{j=1}^{N} K(x_i, x_j)\, w_j \quad \text{for } i = 1, \ldots, N.$$



- Direct computation $0(N^2)$

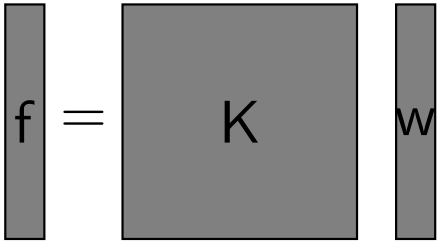


$\mathcal{O}(N^2)$

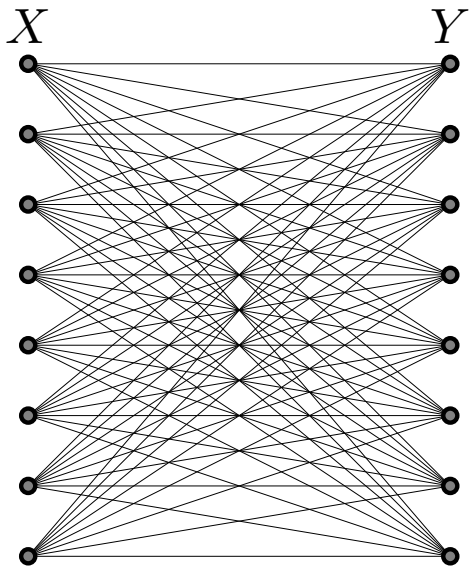
FMM : Motivations

- Pairwise interactions among N bodies.

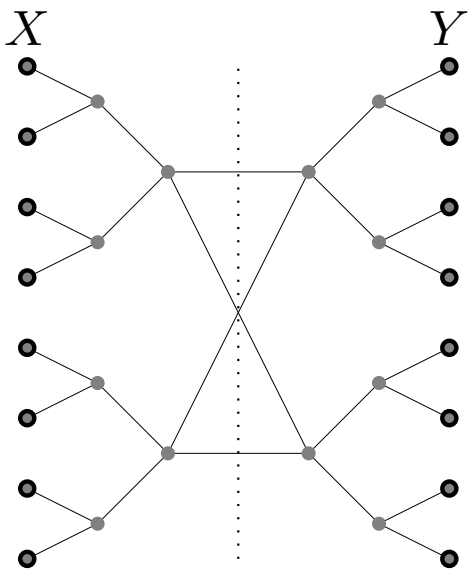
$$f_i = \sum_{j=1}^{N} K(x_i, x_j)\, w_j \quad \text{for } i = 1, \ldots, N.$$



- Direct computation $0(N^2)$ $\rightarrow$ FMM $O(N)$.



$\mathcal{O}(N^2)$



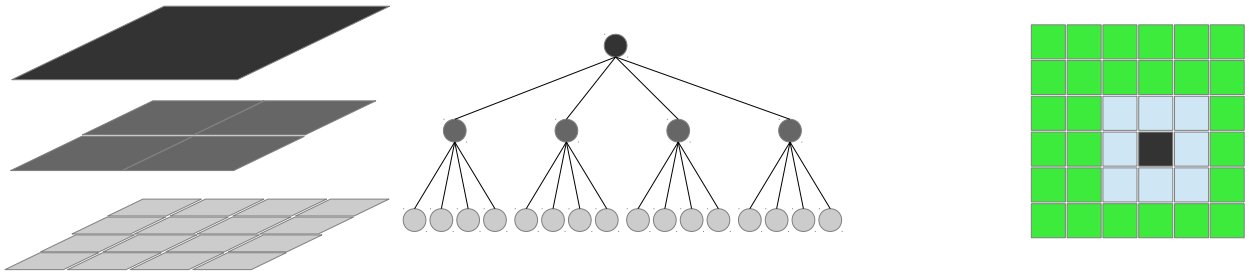
$\mathcal{O}(N)$

FMM : Overview

- Potential decomposition

$$f_i = f_i^{near} + f_i^{far}$$

  - Involve hierarchical space decomposition with an tree (**octree**)



  - Near field is computed by direct interactions.
  - Different approaches to approximate the far field
    - Expansions (Cartesian or Spherical)
    - Interpolation (Chebyshev FMM)

Chebyshev FMM

Idea: Interpolate the kernel on a Chebyshev grid

$$\frac{1}{|\mathbf{x}-\mathbf{y}|} \sim \sum_{m=1}^{\ell} S_\ell(\mathbf{x}, \bar{x}_m) \sum_{n=1}^{\ell} \frac{1}{|\bar{x}_m - \bar{y}_m|} S_\ell(\mathbf{y}, \bar{y}_n)$$

Chebyshev FMM

Idea: Interpolate the kernel on a Chebyshev grid

$$\frac{1}{|\mathbf{x}-\mathbf{y}|} \sim \sum_{m=1}^{\ell} S_\ell(\mathbf{x}, \bar{x}_m) \sum_{n=1}^{\ell} \frac{1}{|\bar{x}_m - \bar{y}_m|} S_\ell(\mathbf{y}, \bar{y}_n)$$

S_ℓ is the interpolation polynomial (the higher ℓ the more accurate is the far field)

$$S_\ell(x, x_m) = \frac{1}{\ell} + \frac{2}{\ell} \sum_{n=1}^{\ell-1} T_n(\bar{x}_m) T_n(x)$$

Chebyshev FMM

Idea: Interpolate the kernel on a Chebyshev grid

$$\frac{1}{|\mathbf{x}-\mathbf{y}|} \sim \sum_{m=1}^{\ell} S_\ell(\mathbf{x}, \bar{x}_m) \sum_{n=1}^{\ell} \frac{1}{|\bar{x}_m - \bar{y}_m|} S_\ell(\mathbf{y}, \bar{y}_n)$$

S_ℓ is the interpolation polynomial (the higher ℓ the more accurate is the far field)

$$S_\ell(x, x_m) = \frac{1}{\ell} + \frac{2}{\ell} \sum_{n=1}^{\ell-1} T_n(\bar{x}_m) T_n(x)$$

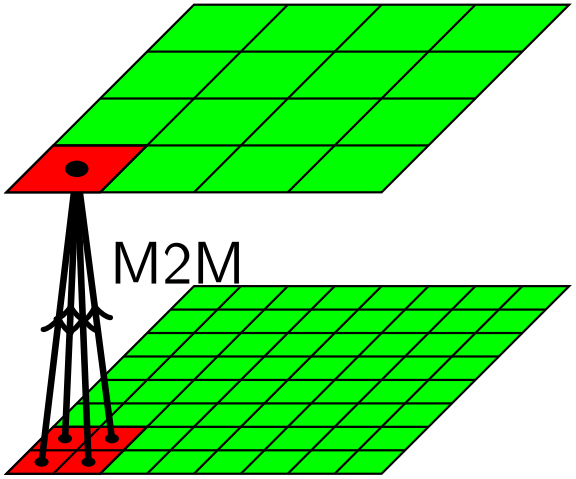
For all $i = 1, \ldots, N$.

$$f_i = \sum_{j=1}^{N} \frac{1}{|x_i - y_j|} w_j$$

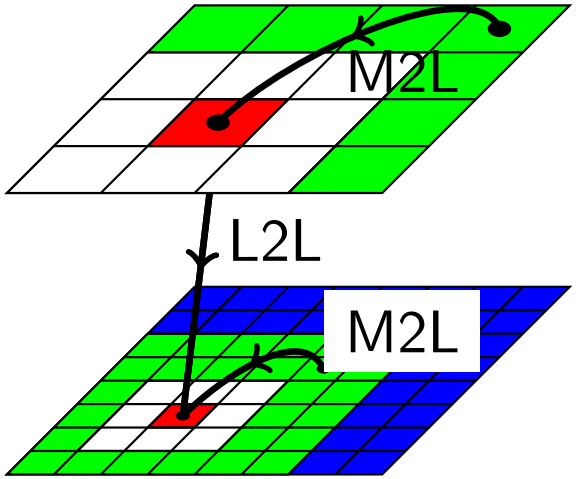
$$\sim \underbrace{\sum_{m=1}^{\ell} S_\ell(x_i, \bar{x}_m)}_{\text{L2L}} \underbrace{\sum_{n=1}^{\ell} \frac{1}{|\bar{x}_m - \bar{y}_m|}}_{\text{M2L}} \underbrace{\sum_{j=1}^{N^{far}} S_\ell(y_j, \bar{y}_n)\, w_j}_{\text{M2M}} + \underbrace{\sum_{j=1}^{N^{near}} \frac{1}{|x_i - y_j|} w_j}_{\text{P2P}}$$

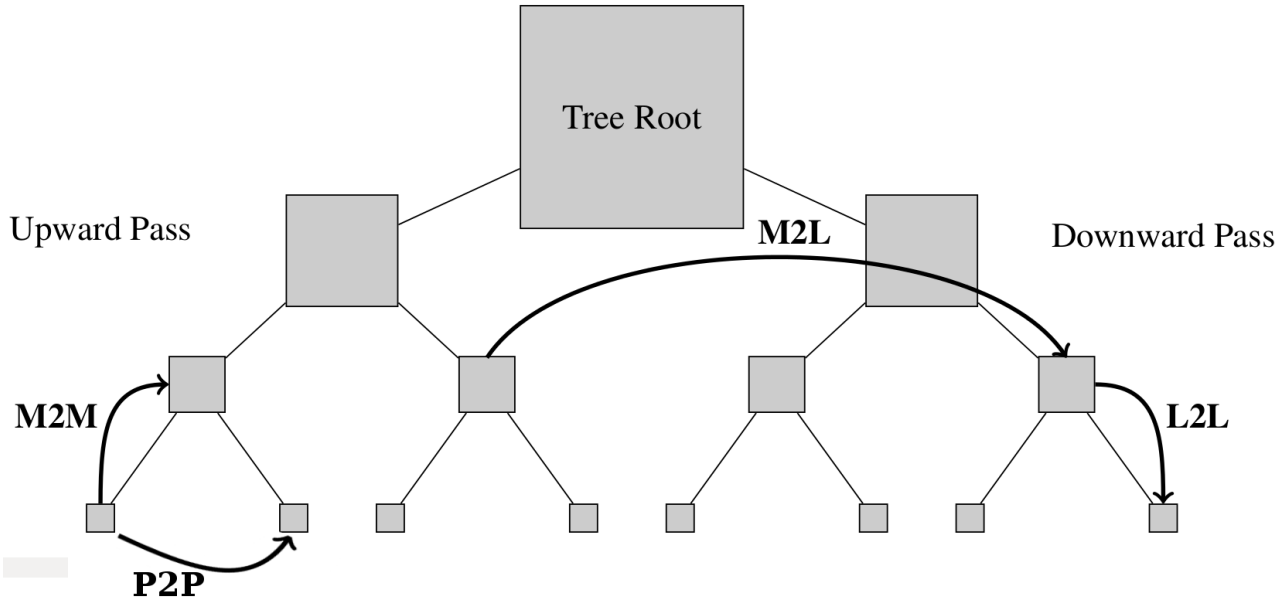
FMM : O(N) algorithm

Upward pass



Downward pass



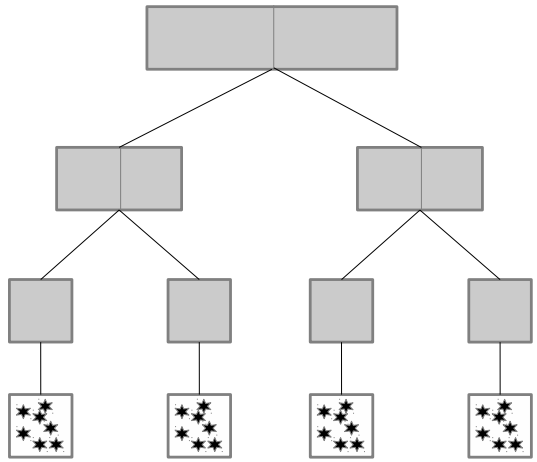


FMM on a runtime

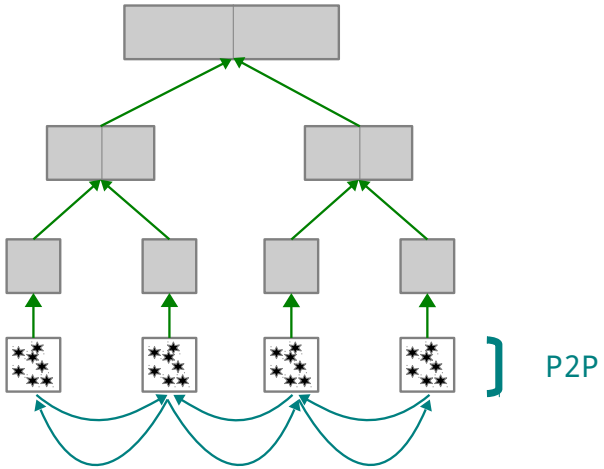
Fork-join parallelization : Description

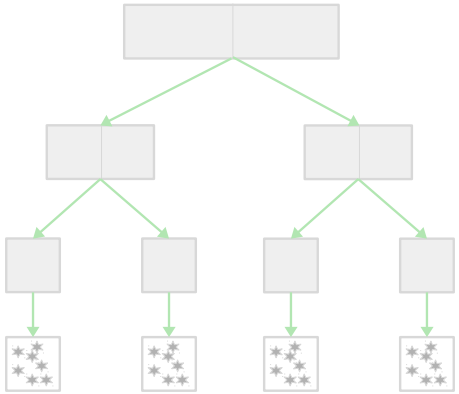
Straightforward parallelization:

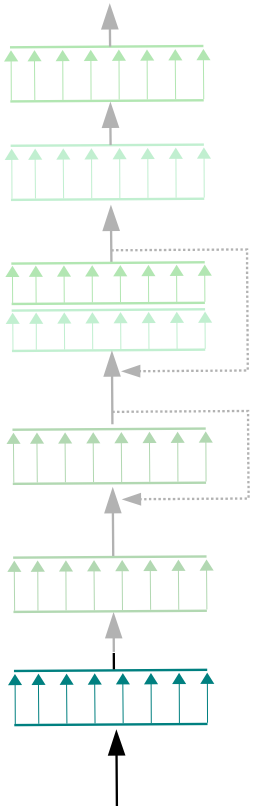
- Using fork-join model
- Progress level by level
- For example using $OpenMP\ for$



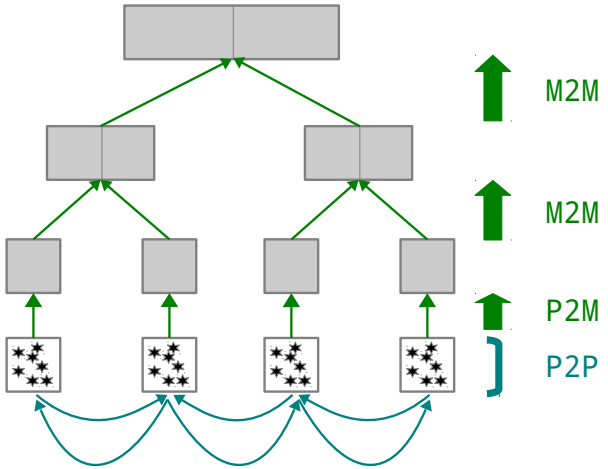
Fork-join parallelization : Simplified illustration

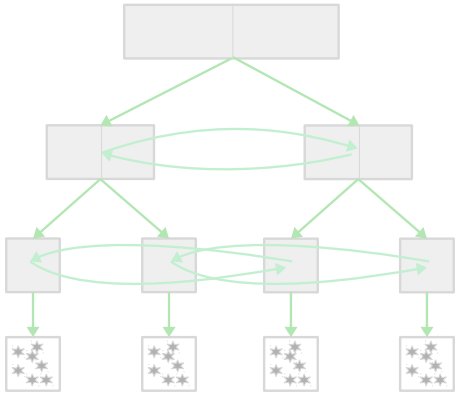


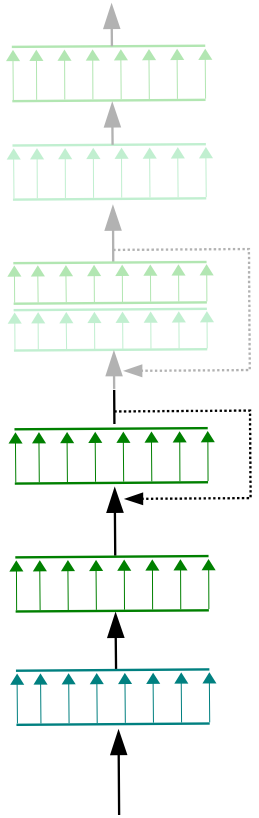




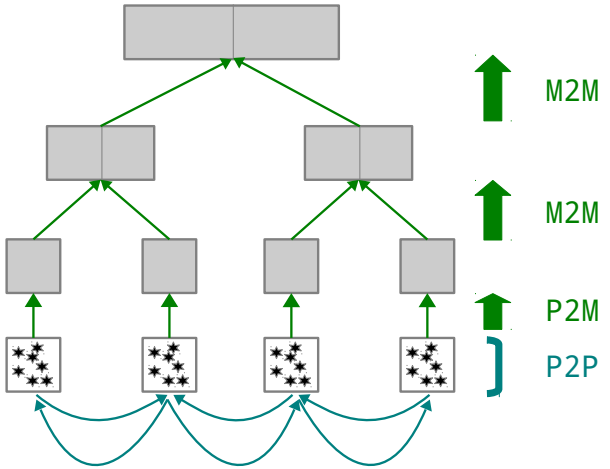
Fork-join parallelization : Simplified illustration

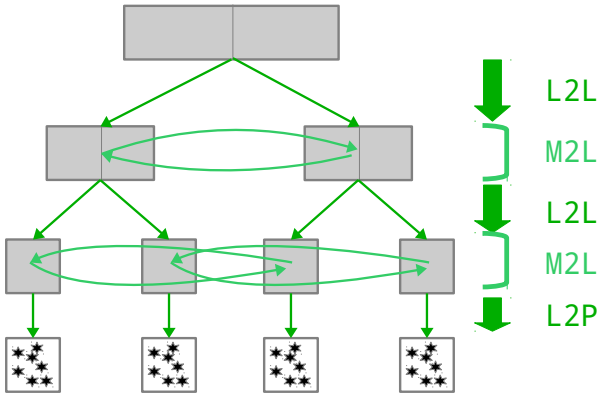


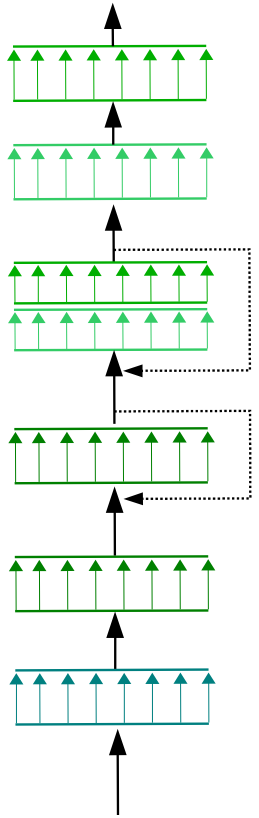




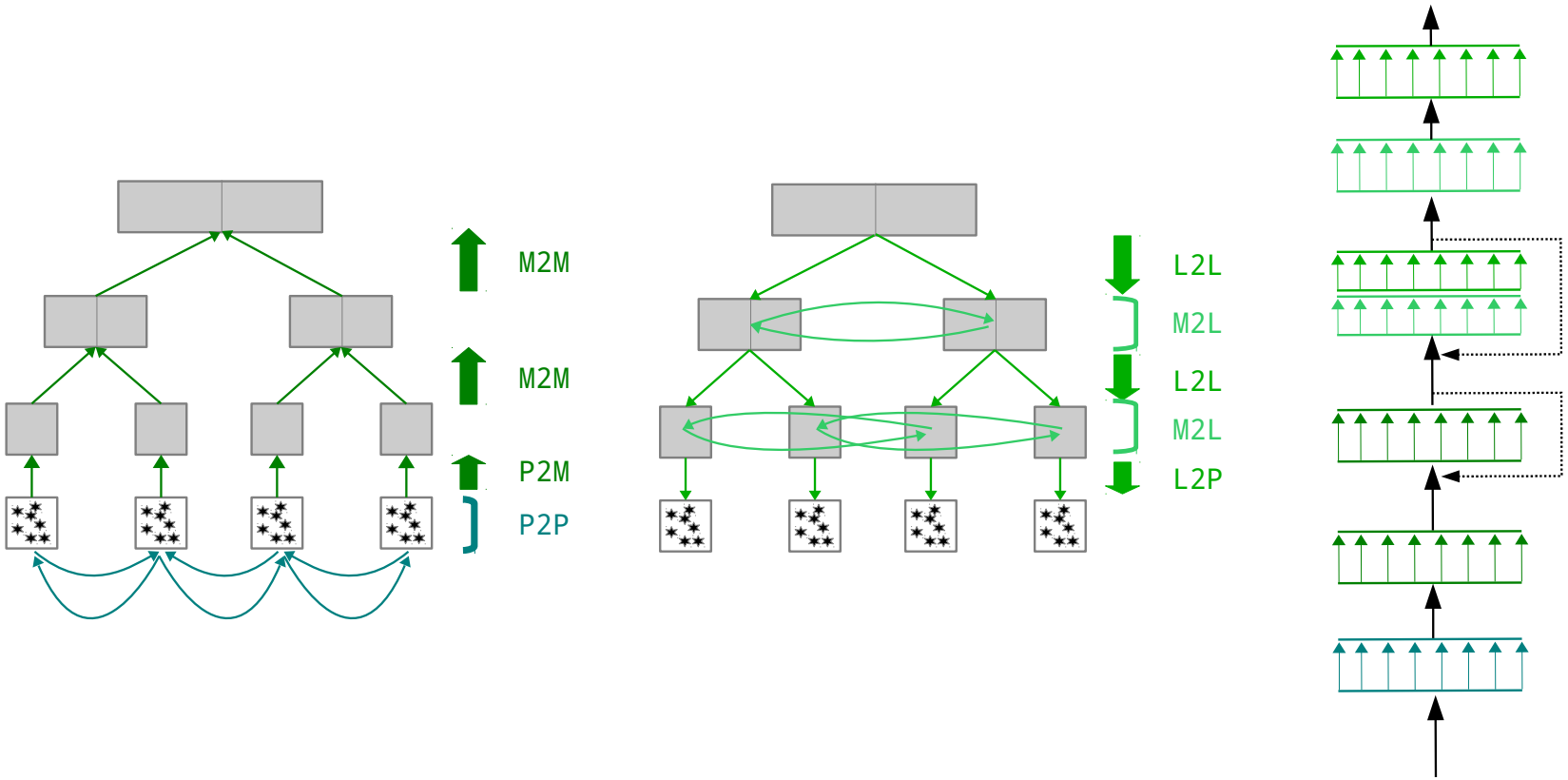
Fork-join parallelization : Simplified illustration





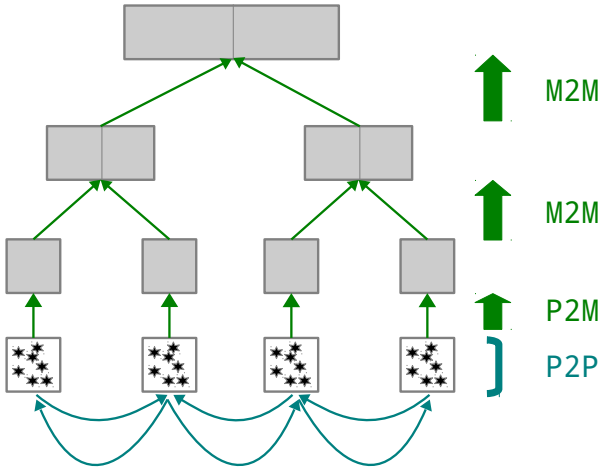


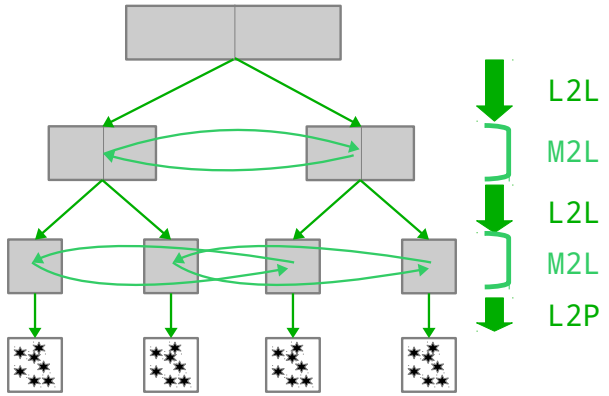
Fork-join parallelization : Simplified illustration

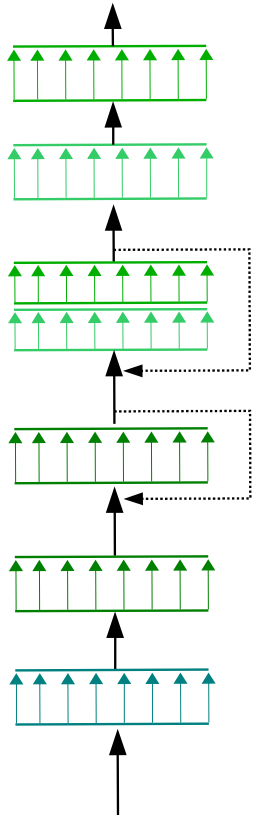


Limits: Synchronizations (+ bottleneck at the top of the tree).

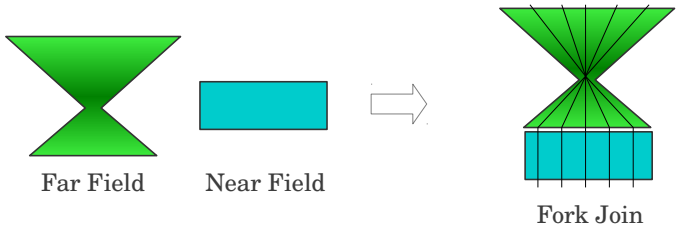
Fork-join parallelization : Simplified illustration



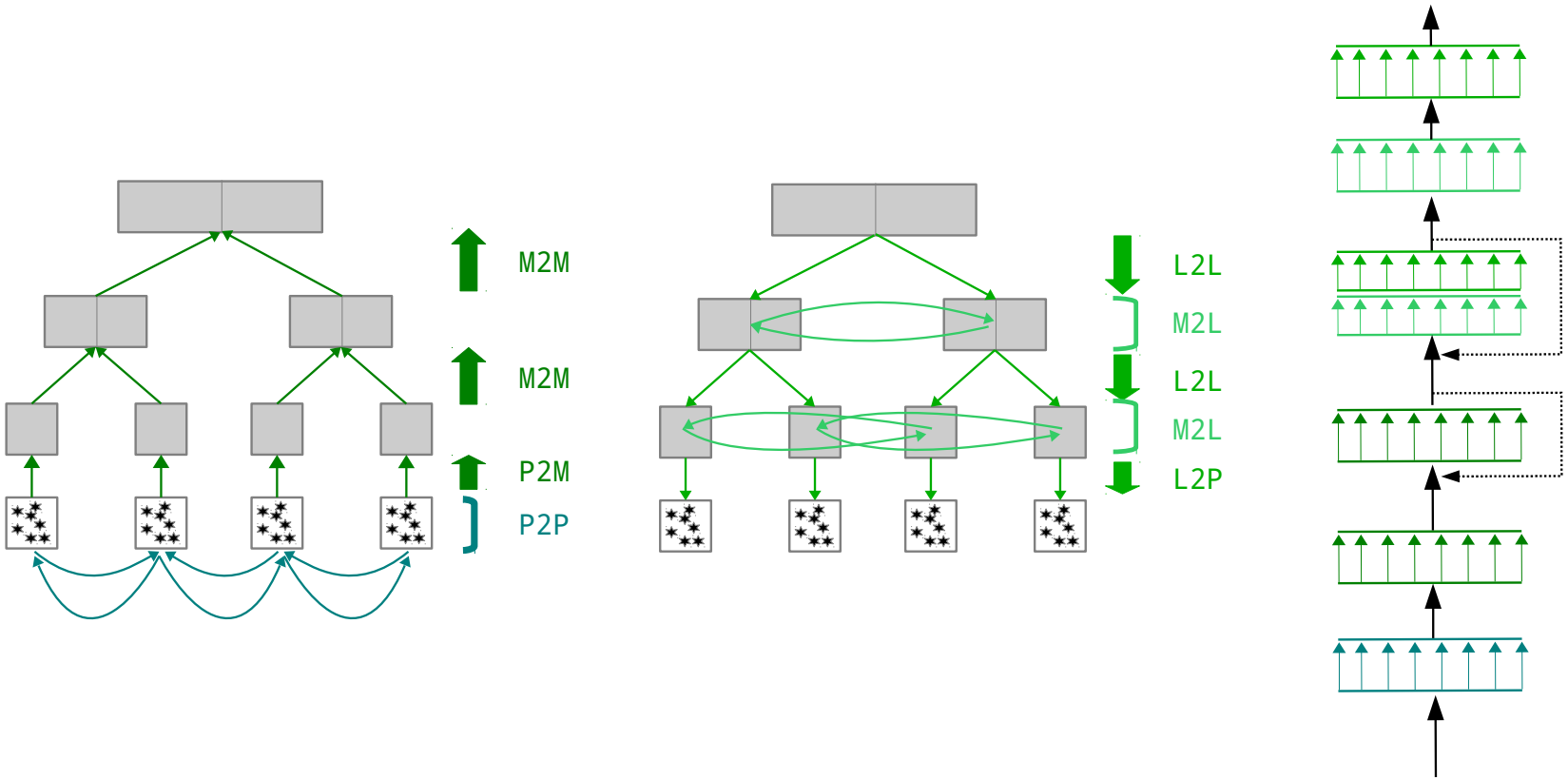




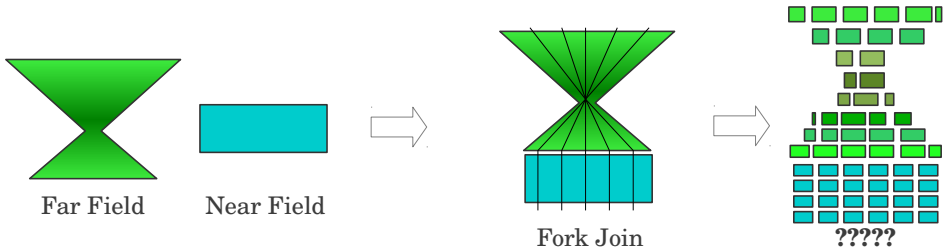
Limits: Synchronizations (+ bottleneck at the top of the tree).



Fork-join parallelization : Simplified illustration



Limits: Synchronizations (+ bottleneck at the top of the tree).



StarPU : Description

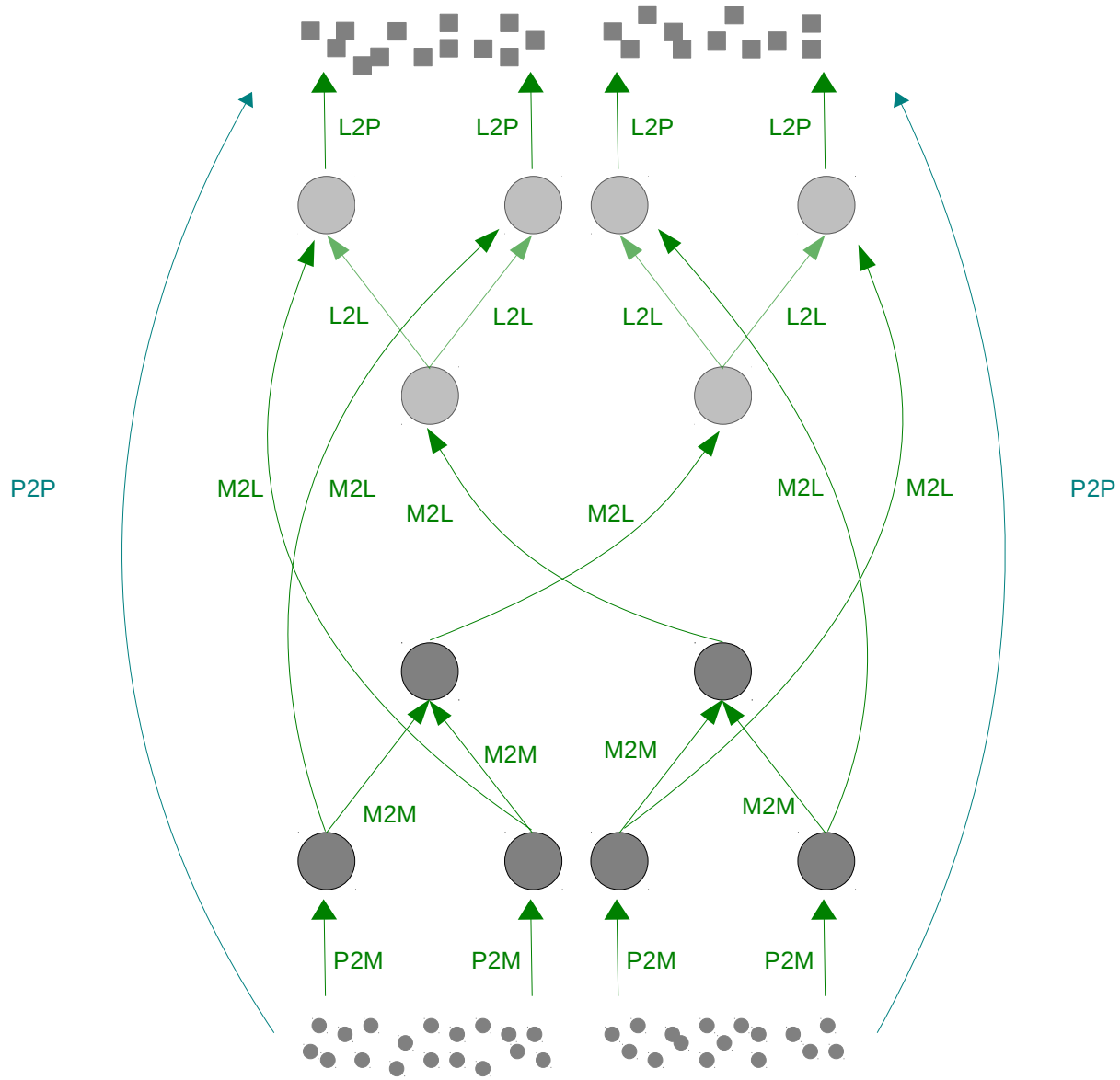
StarPU runtime

- Advanced tasks scheduling (customizable scheduler)
- Tasks dependencies expression
- Gives an abstraction of the architecture (useful for accelerators)
- http://runtime.bordeaux.inria.fr/StarPU/

Insert_task function:

- insert_task (operator, {READ,WRITE}, data, ...)

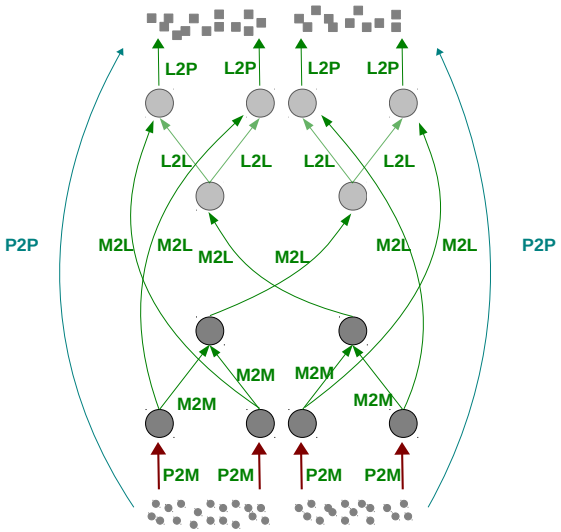
Exploiting the parallelism of the FMM



StarPU : Simplified example

Cells access → cell[level][position]

insert_task(P2M, READ, **particles[0]**, WRITE, **cell[2][0]**)
insert_task(P2M, READ, **particles[1]**, WRITE, **cell[2][1]**)
insert_task(P2M, READ, **particles[2]**, WRITE, **cell[2][2]**)
insert_task(P2M, READ, **particles[3]**, WRITE, **cell[2][3]**)

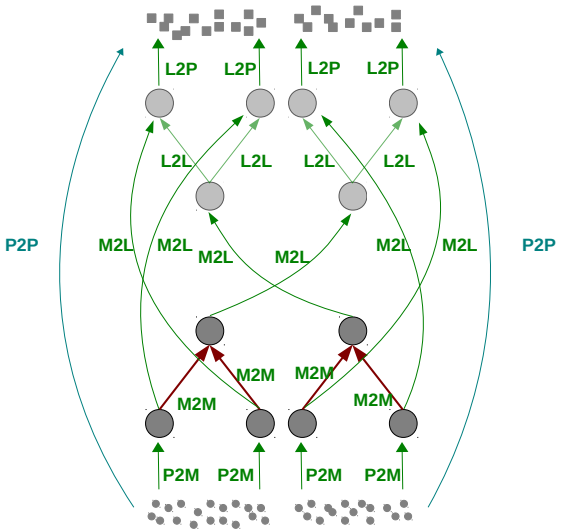


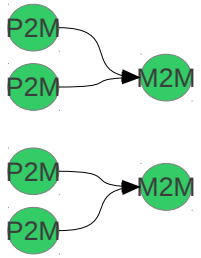
StarPU : Simplified example

Cells access → cell[level][position]

insert_task(P2M, READ, **particles[0]**, WRITE, **cell[2][0]**)
insert_task(P2M, READ, **particles[1]**, WRITE, **cell[2][1]**)
insert_task(P2M, READ, **particles[2]**, WRITE, **cell[2][2]**)
insert_task(P2M, READ, **particles[3]**, WRITE, **cell[2][3]**)

insert_task(M2M, READ, **cell[2][0]**, **cell[2][1]**, WRITE, **cell[1][0]**)
insert_task(M2M, READ, **cell[2][2]**, **cell[2][3]**, WRITE, **cell[1][1]**)





StarPU : Simplified example

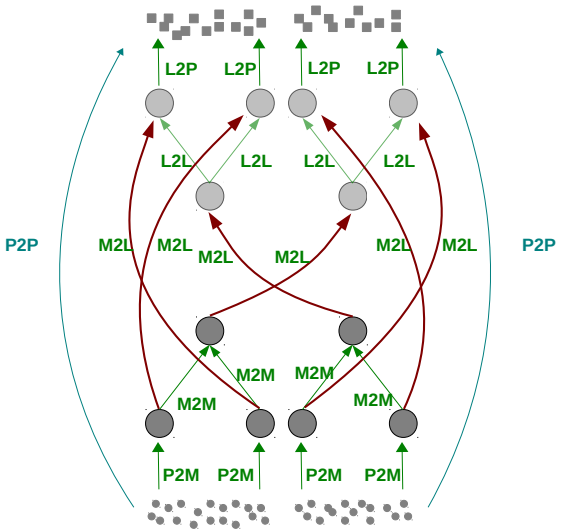
Cells access → cell[level][position]

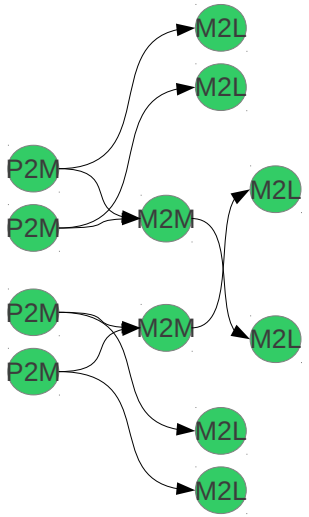
insert_task(P2M, READ, **particles[0]**, WRITE, **cell[2][0]**)
insert_task(P2M, READ, **particles[1]**, WRITE, **cell[2][1]**)
insert_task(P2M, READ, **particles[2]**, WRITE, **cell[2][2]**)
insert_task(P2M, READ, **particles[3]**, WRITE, **cell[2][3]**)

insert_task(M2M, READ, **cell[2][0]**, **cell[2][1]**, WRITE, **cell[1][0]**)
insert_task(M2M, READ, **cell[2][2]**, **cell[2][3]**, WRITE, **cell[1][1]**)

insert_task(M2L, READ, **cell[1][0].interaction**, WRITE, **cell[1][0]**)
insert_task(M2L, READ, **cell[1][1].interaction**, WRITE, **cell[1][1]**)

insert_task(M2L, READ, **cell[2][0].interaction**, WRITE, **cell[2][0]**)
insert_task(M2L, READ, **cell[2][1].interaction**, WRITE, **cell[2][1]**)
insert_task(M2L, READ, **cell[2][2].interaction**, WRITE, **cell[2][2]**)
insert_task(M2L, READ, **cell[2][3].interaction**, WRITE, **cell[2][3]**)





StarPU : Simplified example

Cells access → cell[level][position]

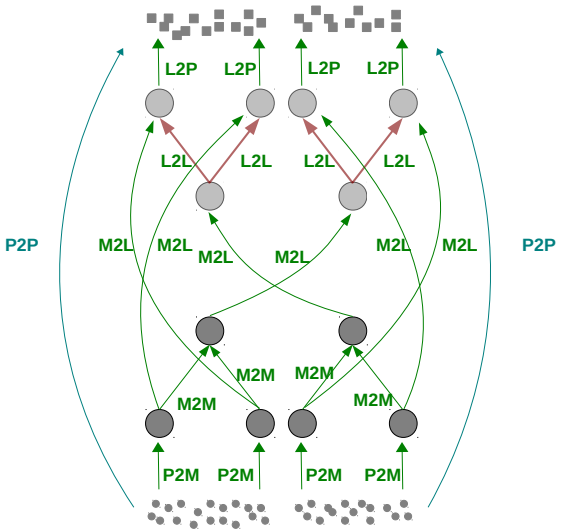
insert_task(P2M, READ, **particles[0]**, WRITE, **cell[2][0]**)
insert_task(P2M, READ, **particles[1]**, WRITE, **cell[2][1]**)
insert_task(P2M, READ, **particles[2]**, WRITE, **cell[2][2]**)
insert_task(P2M, READ, **particles[3]**, WRITE, **cell[2][3]**)

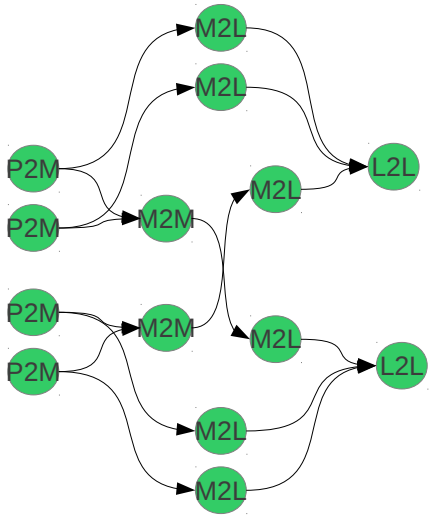
insert_task(M2M, READ, **cell[2][0]**, **cell[2][1]**, WRITE, **cell[1][0]**)
insert_task(M2M, READ, **cell[2][2]**, **cell[2][3]**, WRITE, **cell[1][1]**)

insert_task(M2L, READ, **cell[1][0].interaction**, WRITE, **cell[1][0]**)
insert_task(M2L, READ, **cell[1][1].interaction**, WRITE, **cell[1][1]**)

insert_task(M2L, READ, **cell[2][0].interaction**, WRITE, **cell[2][0]**)
insert_task(M2L, READ, **cell[2][1].interaction**, WRITE, **cell[2][1]**)
insert_task(M2L, READ, **cell[2][2].interaction**, WRITE, **cell[2][2]**)
insert_task(M2L, READ, **cell[2][3].interaction**, WRITE, **cell[2][3]**)

insert_task(L2L, READ, **cell[1][0]**, WRITE, **cell[2][0]**, **cell[2][1]**)
insert_task(L2L, READ, **cell[1][1]**, WRITE, **cell[2][2]**, **cell[2][3]**)

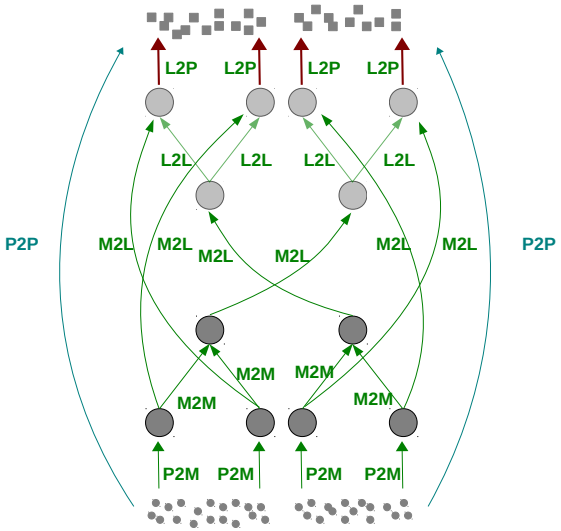


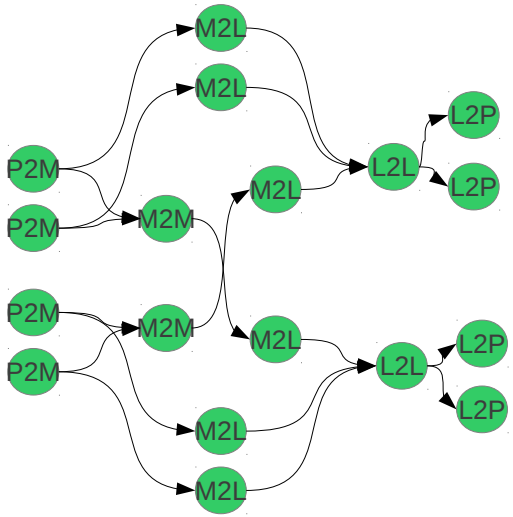


StarPU : Simplified example

Cells access → cell[level][position]

```
insert_task(P2M, READ, particles[0], WRITE, cell[2][0])
insert_task(P2M, READ, particles[1], WRITE, cell[2][1])
insert_task(P2M, READ, particles[2], WRITE, cell[2][2])
insert_task(P2M, READ, particles[3], WRITE, cell[2][3])

insert_task(M2M, READ, cell[2][0], cell[2][1], WRITE, cell[1][0])
insert_task(M2M, READ, cell[2][2], cell[2][3], WRITE, cell[1][1])

insert_task(M2L, READ, cell[1][0].interaction, WRITE, cell[1][0])
insert_task(M2L, READ, cell[1][1].interaction, WRITE, cell[1][1])

insert_task(M2L, READ, cell[2][0].interaction, WRITE, cell[2][0])
insert_task(M2L, READ, cell[2][1].interaction, WRITE, cell[2][1])
insert_task(M2L, READ, cell[2][2].interaction, WRITE, cell[2][2])
insert_task(M2L, READ, cell[2][3].interaction, WRITE, cell[2][3])

insert_task(L2L, READ, cell[1][0], WRITE, cell[2][0], cell[2][1])
insert_task(L2L, READ, cell[1][1], WRITE, cell[2][2], cell[2][3])

insert_task(L2P, READ, cell[2][0], WRITE, particles[0])
insert_task(L2P, READ, cell[2][1], WRITE, particles[1])
insert_task(L2P, READ, cell[2][2], WRITE, particles[2])
insert_task(L2P, READ, cell[2][3], WRITE, particles[3])
```





StarPU : Simplified example

Cells access → cell[level][position]

insert_task(P2M, READ, **particles[0]**, WRITE, **cell[2][0]**)
insert_task(P2M, READ, **particles[1]**, WRITE, **cell[2][1]**)
insert_task(P2M, READ, **particles[2]**, WRITE, **cell[2][2]**)
insert_task(P2M, READ, **particles[3]**, WRITE, **cell[2][3]**)

insert_task(M2M, READ, **cell[2][0]**, **cell[2][1]**, WRITE, **cell[1][0]**)
insert_task(M2M, READ, **cell[2][2]**, **cell[2][3]**, WRITE, **cell[1][1]**)

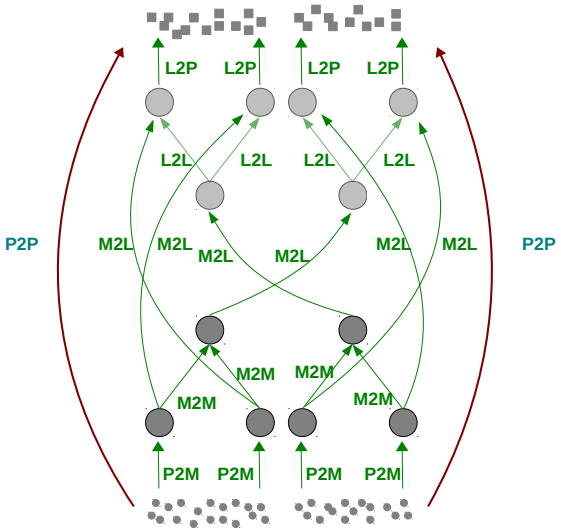
insert_task(M2L, READ, **cell[1][0].interaction**, WRITE, **cell[1][0]**)
insert_task(M2L, READ, **cell[1][1].interaction**, WRITE, **cell[1][1]**)

insert_task(M2L, READ, **cell[2][0].interaction**, WRITE, **cell[2][0]**)
insert_task(M2L, READ, **cell[2][1].interaction**, WRITE, **cell[2][1]**)
insert_task(M2L, READ, **cell[2][2].interaction**, WRITE, **cell[2][2]**)
insert_task(M2L, READ, **cell[2][3].interaction**, WRITE, **cell[2][3]**)

insert_task(L2L, READ, **cell[1][0]**, WRITE, **cell[2][0]**, **cell[2][1]**)
insert_task(L2L, READ, **cell[1][1]**, WRITE, **cell[2][2]**, **cell[2][3]**)

insert_task(L2P, READ, **cell[2][0]**, WRITE, **particles[0]**)
insert_task(L2P, READ, **cell[2][1]**, WRITE, **particles[1]**)
insert_task(L2P, READ, **cell[2][2]**, WRITE, **particles[2]**)
insert_task(L2P, READ, **cell[2][3]**, WRITE, **particles[3]**)

insert_task(P2P, READ, **neighbours[0]**, WRITE, **particles[0]**)
insert_task(P2P, READ, **neighbours[1]**, WRITE, **particles[1]**)
insert_task(P2P, READ, **neighbours[2]**, WRITE, **particles[2]**)
insert_task(P2P, READ, **neighbours[3]**, WRITE, **particles[3]**)



StarPU : Simplified example

Cells access → cell[level][position]

insert_task(P2M, READ, **particles[0]**, WRITE, **cell[2][0]**)
insert_task(P2M, READ, **particles[1]**, WRITE, **cell[2][1]**)
insert_task(P2M, READ, **particles[2]**, WRITE, **cell[2][2]**)
insert_task(P2M, READ, **particles[3]**, WRITE, **cell[2][3]**)

insert_task(M2M, READ, **cell[2][0]**, **cell[2][1]**, WRITE, **cell[1][0]**)
insert_task(M2M, READ, **cell[2][2]**, **cell[2][3]**, WRITE, **cell[1][1]**)

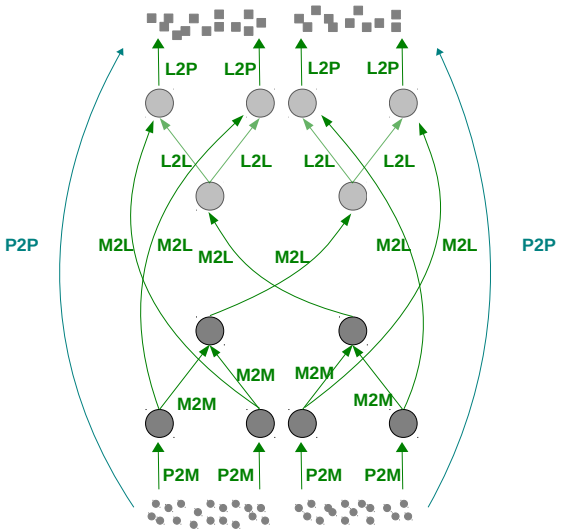
insert_task(M2L, READ, **cell[1][0].interaction**, WRITE, **cell[1][0]**)
insert_task(M2L, READ, **cell[1][1].interaction**, WRITE, **cell[1][1]**)

insert_task(M2L, READ, **cell[2][0].interaction**, WRITE, **cell[2][0]**)
insert_task(M2L, READ, **cell[2][1].interaction**, WRITE, **cell[2][1]**)
insert_task(M2L, READ, **cell[2][2].interaction**, WRITE, **cell[2][2]**)
insert_task(M2L, READ, **cell[2][3].interaction**, WRITE, **cell[2][3]**)

insert_task(L2L, READ, **cell[1][0]**, WRITE, **cell[2][0]**, **cell[2][1]**)
insert_task(L2L, READ, **cell[1][1]**, WRITE, **cell[2][2]**, **cell[2][3]**)

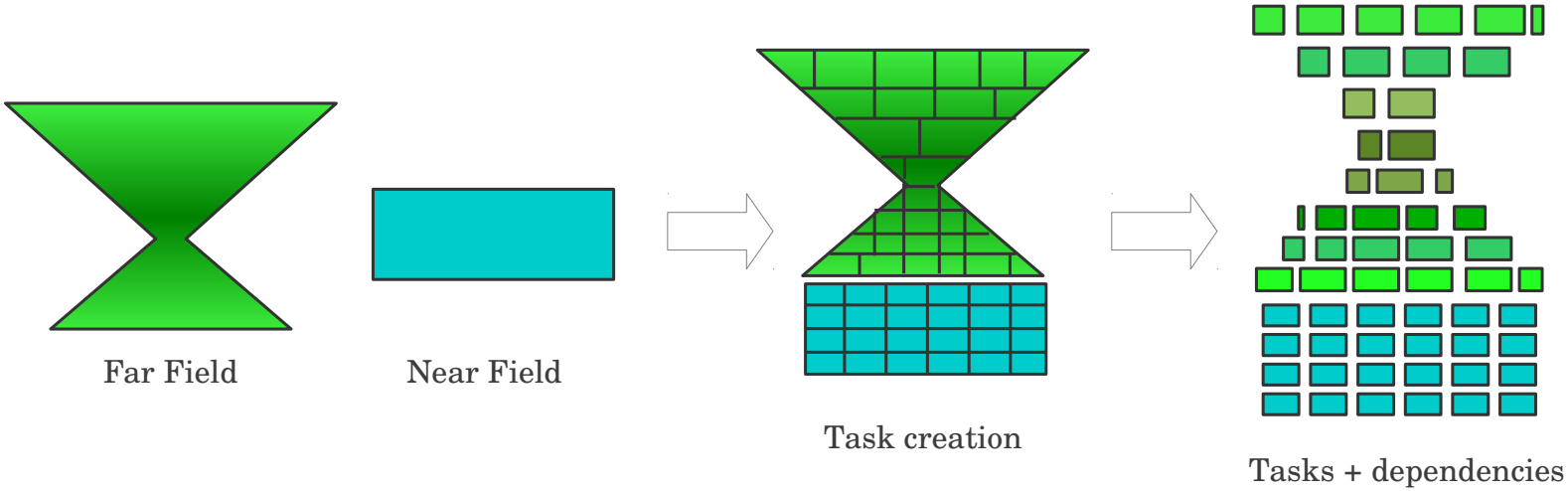
insert_task(P2P, READ, **neighbours[0]**, WRITE, **particles[0]**)
insert_task(P2P, READ, **neighbours[1]**, WRITE, **particles[1]**)
insert_task(P2P, READ, **neighbours[2]**, WRITE, **particles[2]**)
insert_task(P2P, READ, **neighbours[3]**, WRITE, **particles[3]**)

insert_task(L2P, READ, **cell[2][0]**, WRITE, **particles[0]**)
insert_task(L2P, READ, **cell[2][1]**, WRITE, **particles[1]**)
insert_task(L2P, READ, **cell[2][2]**, WRITE, **particles[2]**)
insert_task(L2P, READ, **cell[2][3]**, WRITE, **particles[3]**)



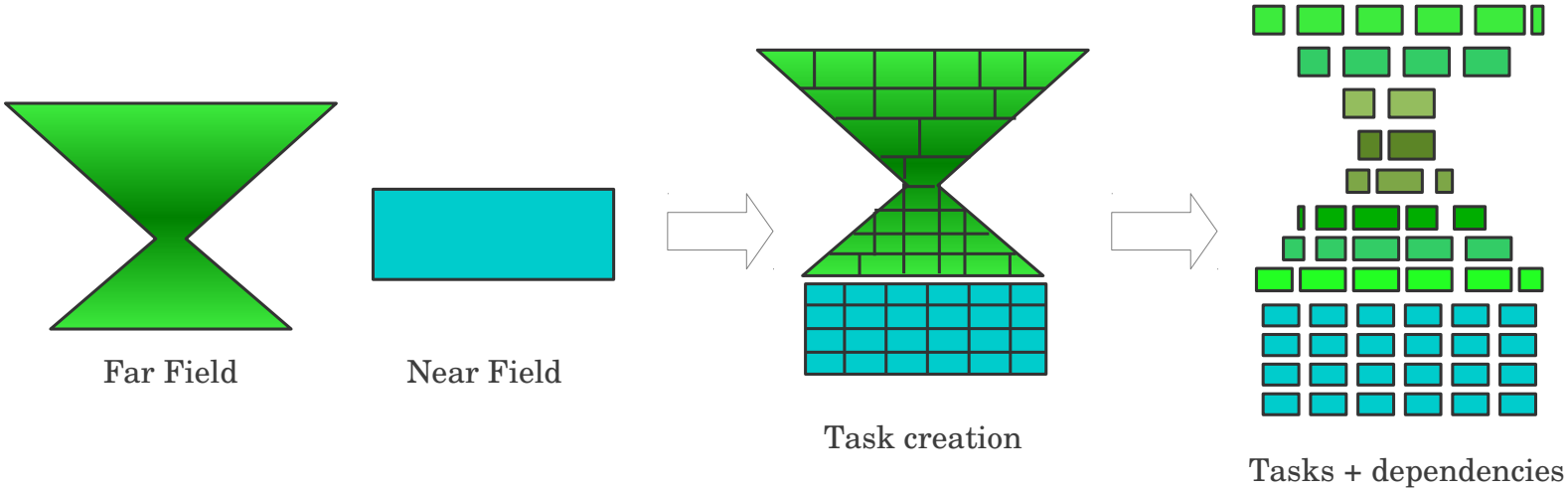
StarPU : Tasks creation and execution

Creation of the tasks using insert function:

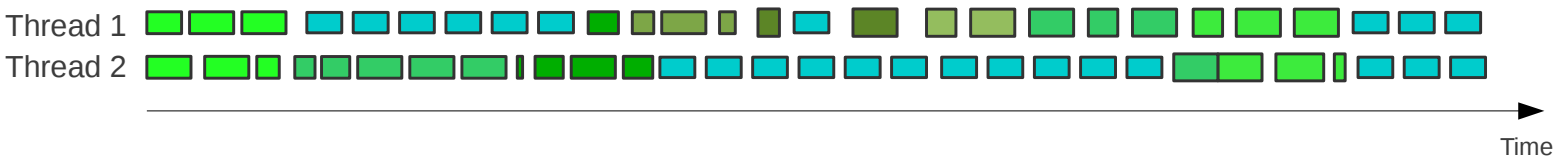


StarPU : Tasks creation and execution

Creation of the tasks using insert function:



Execution of the tasks (trace):



Execution with 2 threads

StarPU : Using GPU kernels

Two GPU kernels : P2P and M2L

- P2P & M2L represent more than 90% of the work

Scheduling:

- Schedulers assign tasks to processing units.
- StarPU proposes several schedulers (but none fits our needs).
- We developed a new scheduler:
  - Based on Eager
  - Enable setting priorities to operators
  - Different priorities for GPU or CPU

Results & Performance

Experimental Setup

ScalFMM:

- Developed in C++ (OpenMP, MPI, and now StarPU)
- Available at http://scalfmm-public.gforge.inria.fr/

Test cluster:

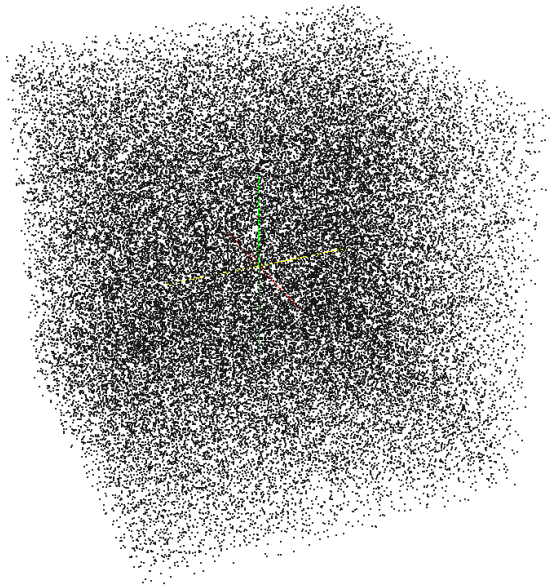
- Heterogeneous 12 CPU and 3 GPU
- CPU dual-socket hexa-core Intel X5650 Nehalem (2.67 GHz)
- GPU Nvidia M2070 (or M2090)

In the following results Intel Compiler (12.0.5) is used.

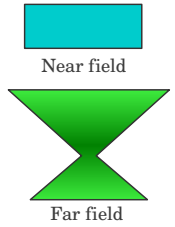
Test case

- Uniform distribution of particles in 3D inside a cube

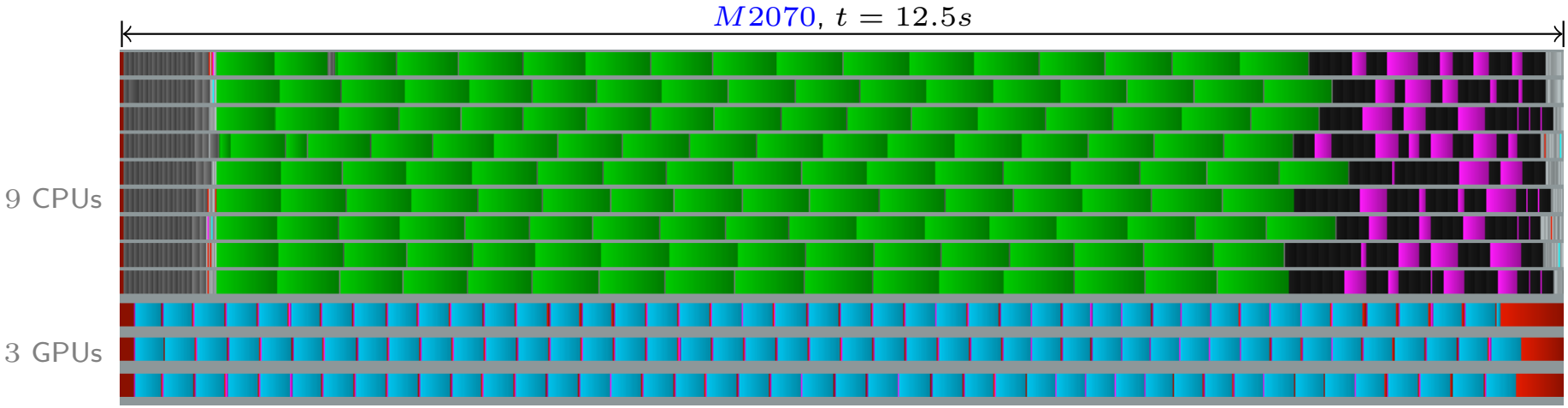
- Particles 30.10^6
- Granularity $n_g = 2000$
- Octree height $h = \{6, 7\}$
- $Acc(\ell) = 10^{-\ell}$, with $\ell = \{2 - 7\}$
- $GPU = \{0, 1, 2, 3\}$



StarPU Execution traces M2070-M2090

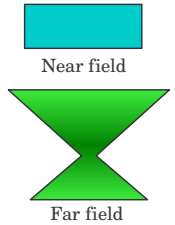


Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$, $Acc(\ell = 5) = 10^{-5}$

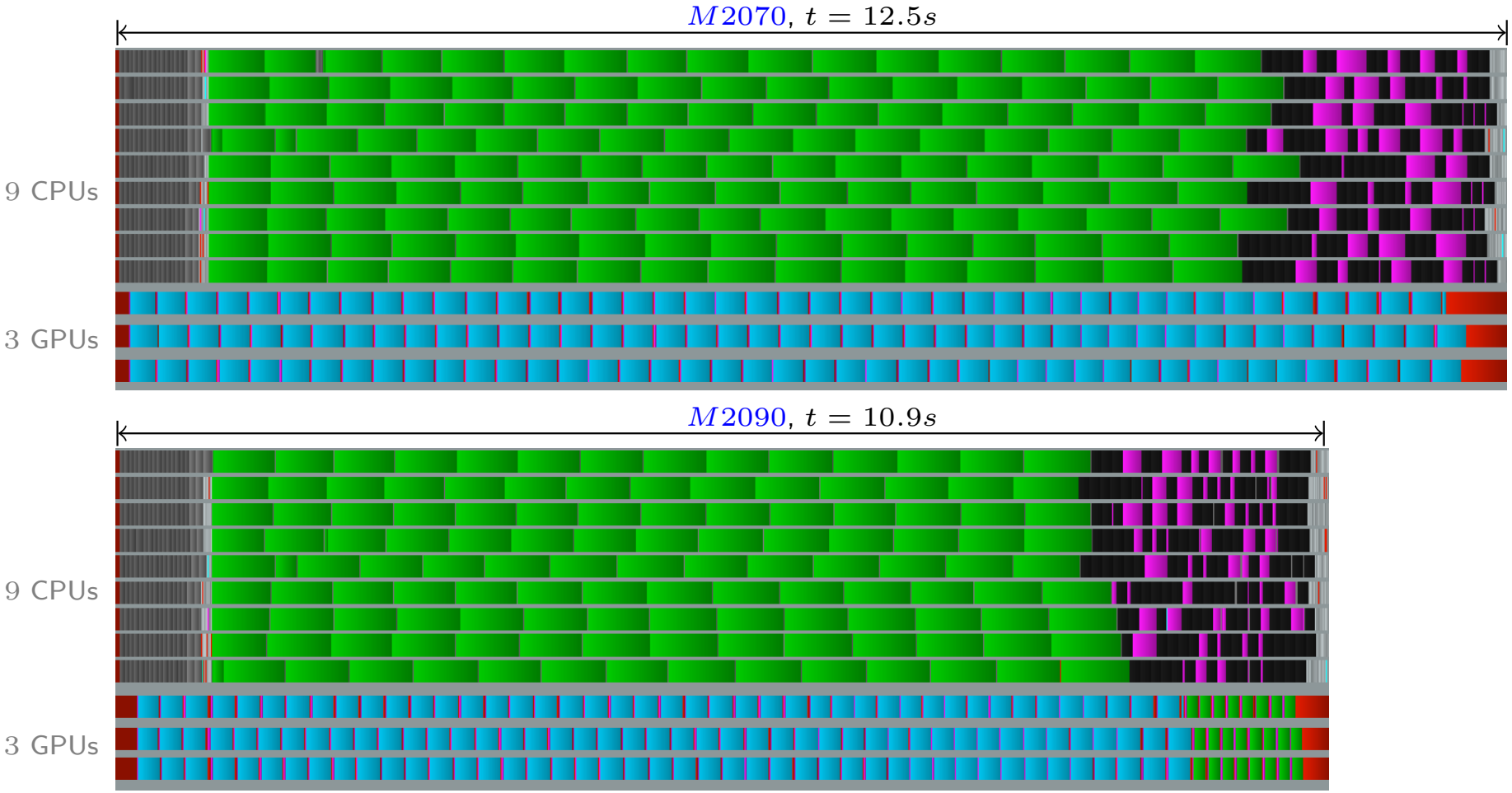


Legend : **P2P**, **P2M**, **M2M**, **M2L**, **L2L**, **L2P**, **Idle**, **Moving data** .

StarPU Execution traces M2070-M2090



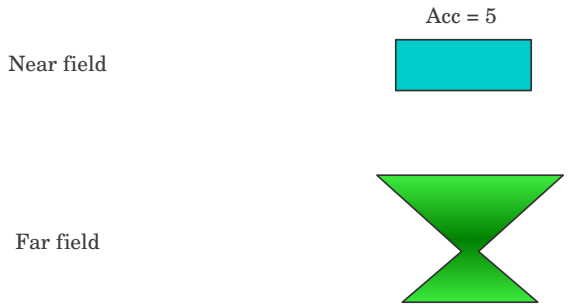
Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$, $Acc(\ell = 5) = 10^{-5}$



Legend : P2P, P2M, M2M, M2L, L2L, L2P, Idle, Moving data .

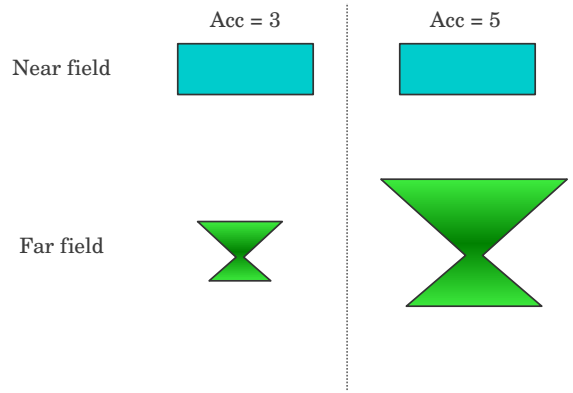
StarPU Execution traces M2070

Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$



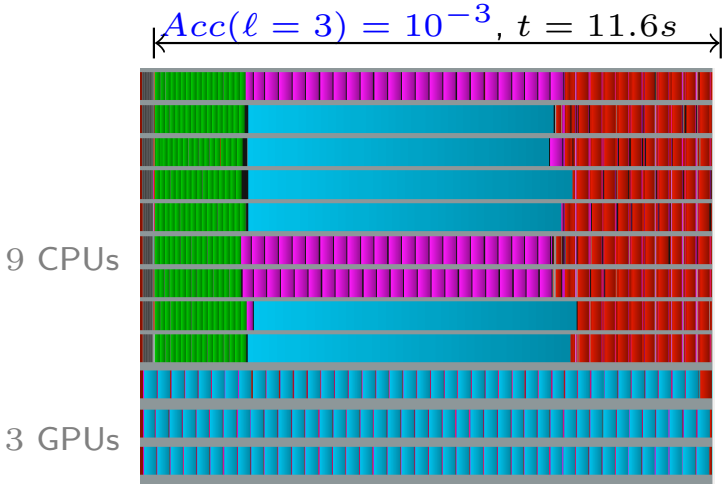
StarPU Execution traces M2070

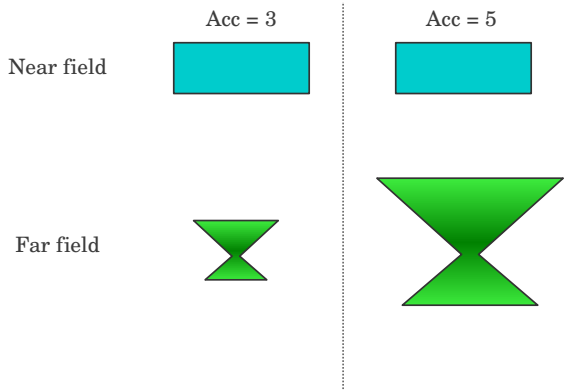
Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$



StarPU Execution traces M2070

Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$

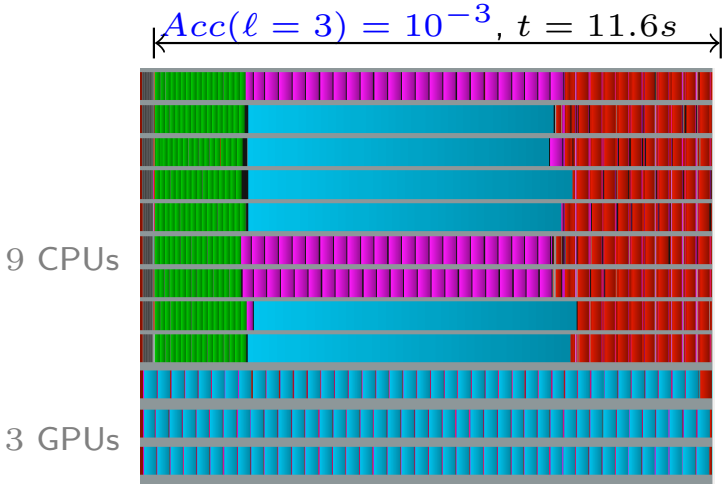


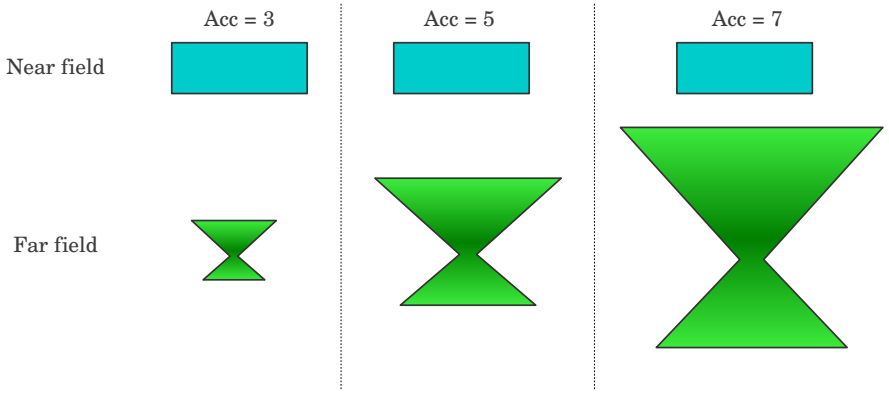


Legend : **P2P**, **P2M**, **M2M**, **M2L**, **L2L**, **L2P**, **Idle**, **Moving data** .

StarPU Execution traces M2070

Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$

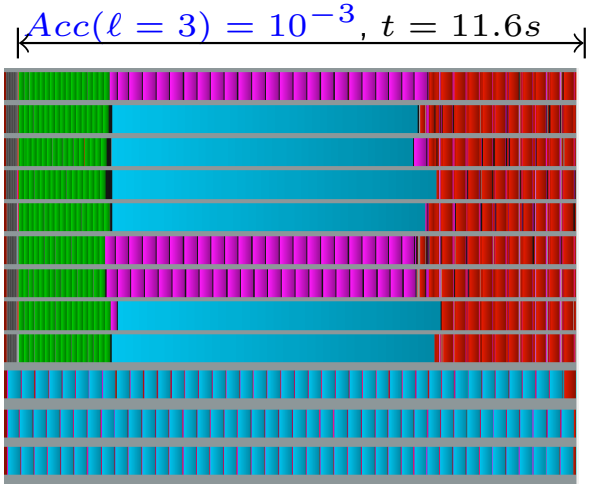


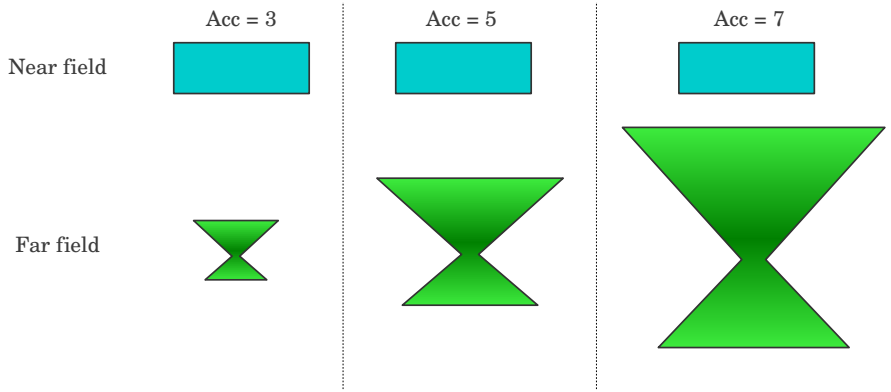


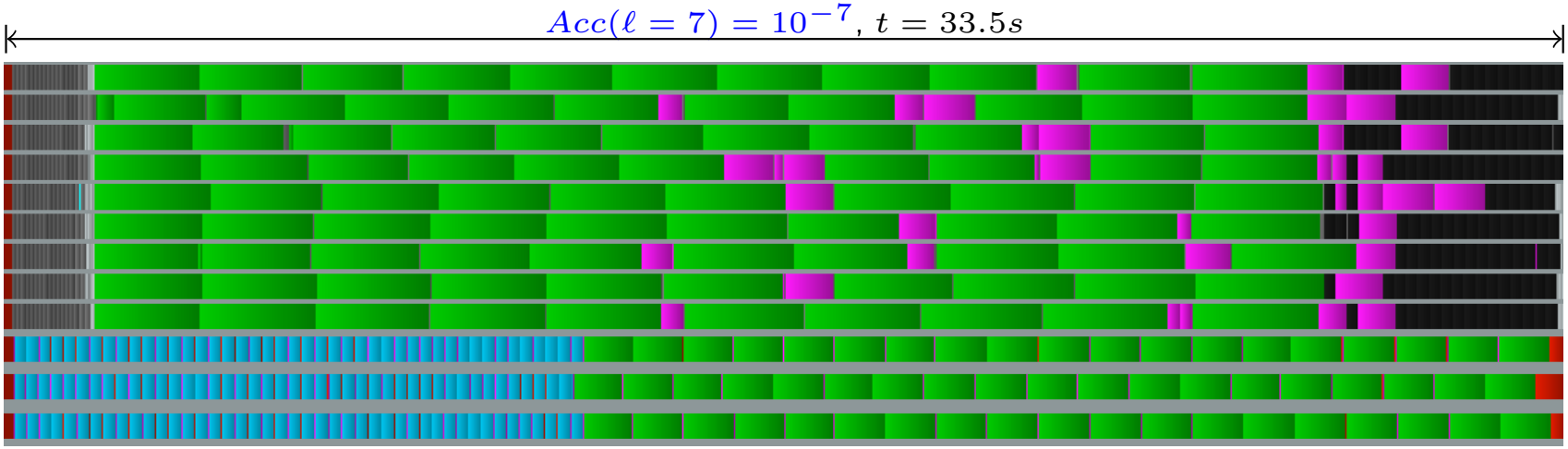
Legend : **P2P**, **P2M**, **M2M**, **M2L**, **L2L**, **L2P**, **Idle**, **Moving data** .

StarPU Execution traces M2070

Particles 30.10^6, Octree height $h = 6$, Granularity $n_g = 2000$



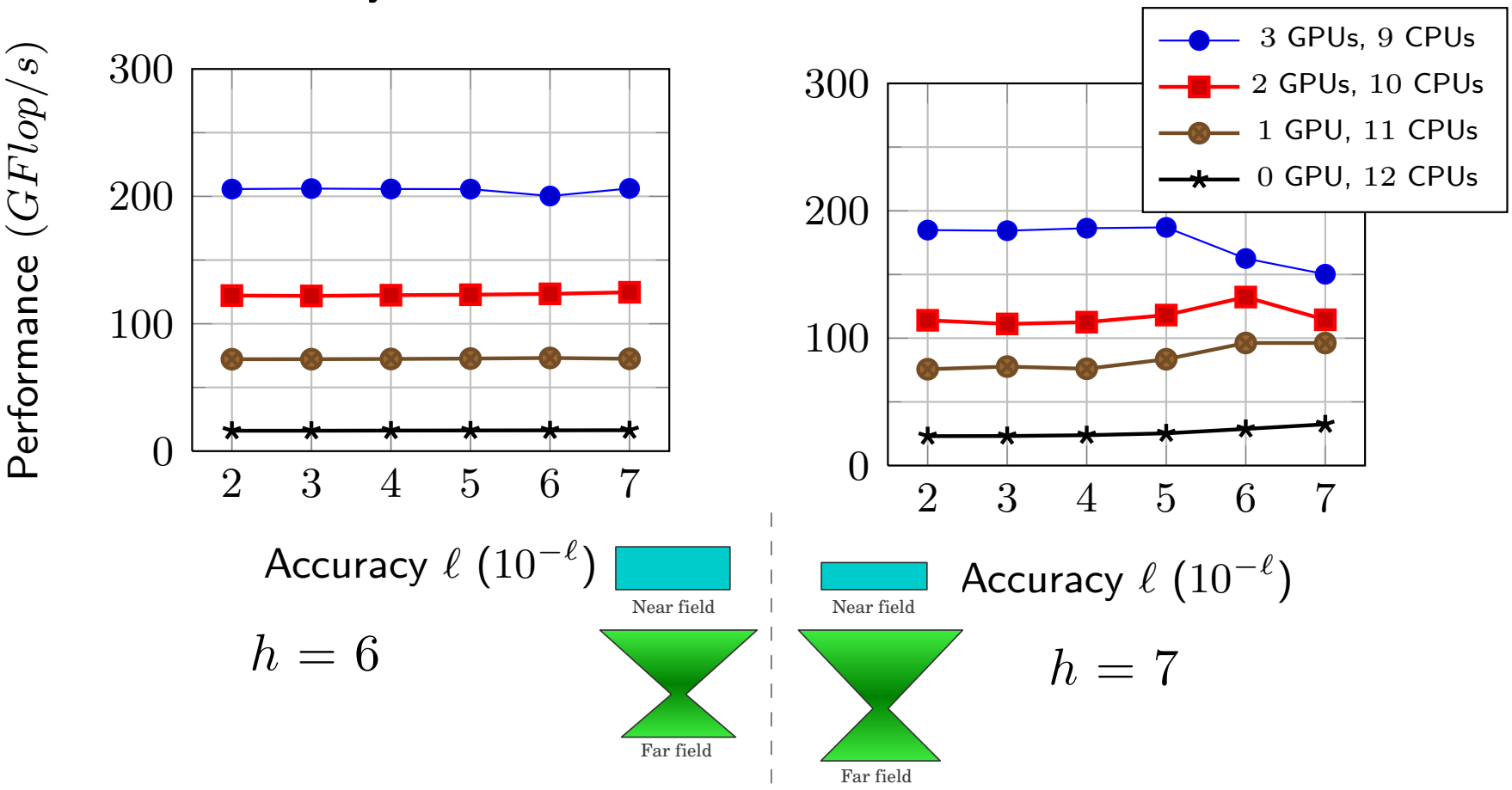




Legend : P2P, P2M, M2M, M2L, L2L, L2P, Idle, Moving data .

Results M2070

30.10^6 particles inside a box, granularity $n_g = 2000$, for 0 to 3 GPU and accuracy from 2 to 7.



Conclusion

- Runtime optimization of parallel mapping was demonstrated.
- Modern heterogeneous computing platforms lead to hard DAG scheduling.
- Approach like Fork-join may be insufficient.
- Code is hardware “independent.” Schedulers determine the best processor to execute a given task.

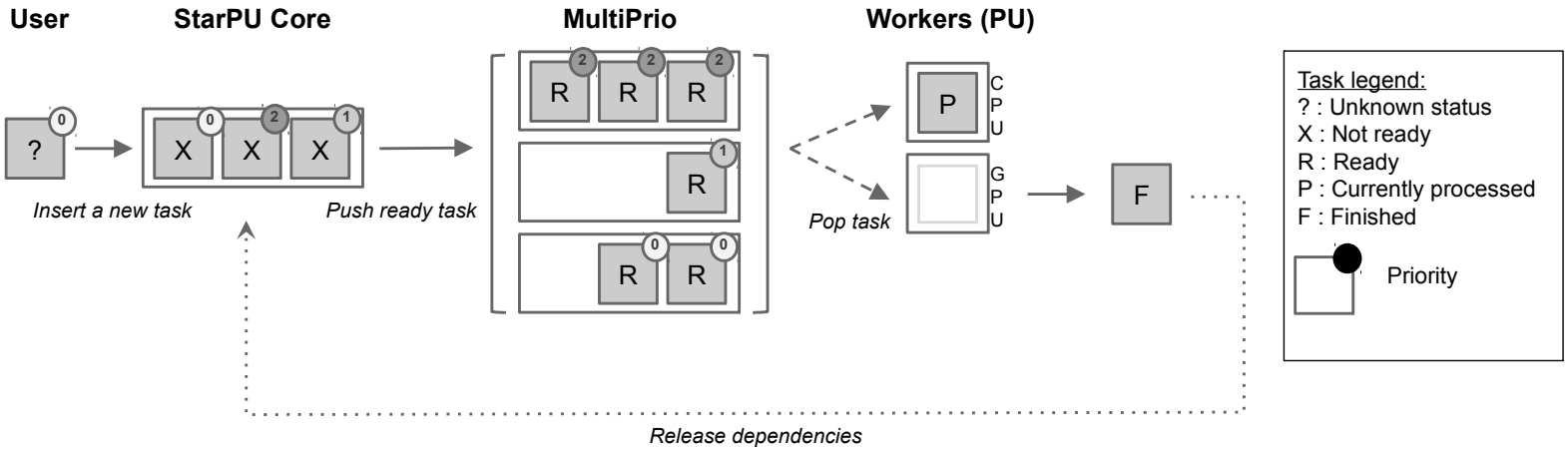
Acknowledgement

- Inria & FastLA associate team.
- StarPU development team.
- ScalFMM http://scalfmm-public.gforge.inria.fr/

Thanks - Questions?

Annexes

Annexe : Scheduler



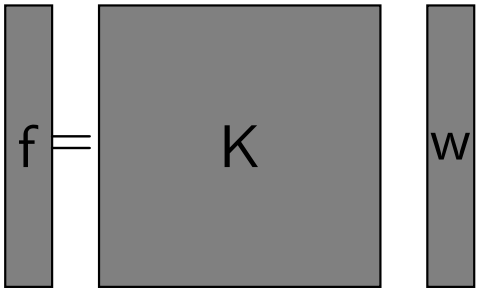
Annexe : Using CPU and GPU

```
function M2L()
    forall the levels lv from 2 to leaf_level do
        foreach Block bl in blocks[lv] do
            insert_task( cpu_m2l, gpu_m2l, READ, bl.interactions,
            WRITE, bl);
```

- StarPU is able to compute a M2L on the CPU or on the device (after moving the data)

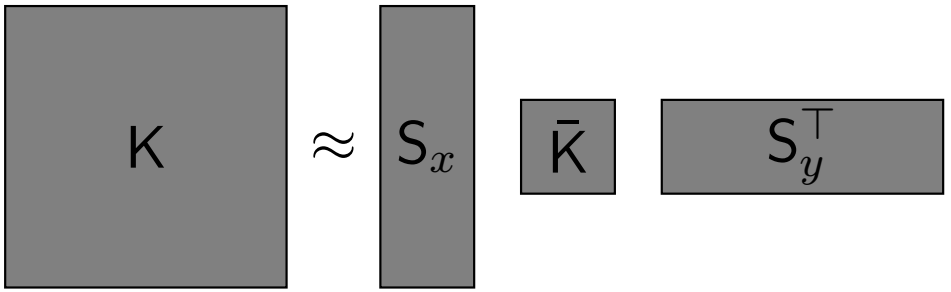
Chebyshev FMM : Direct Matrix Vector

- $\mathsf{f} = \mathsf{K}\,\mathsf{w}$
- $(\mathsf{K})_{ij} = K(x_i, y_j) = \frac{1}{|x_i - y_j|}$



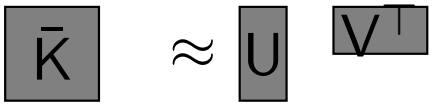
Interpolation based Matrix Vector (low rank approximation)

- $\mathsf{K} \approx \mathsf{S}_x\,\bar{\mathsf{K}}\,\mathsf{S}_y^T$
- $\mathsf{S}_x, \mathsf{S}_y\; dim\; \mathsf{N} \times \ell^3$
- $\bar{\mathsf{K}}\; dim\; \ell^3 \times \ell^3$



Second low rank approximation (SVD)

- $\bar{\mathsf{K}} \approx \mathsf{U}\,\mathsf{V}^\top$
- $\mathsf{U}, \mathsf{V}\; dim\; \ell^3 \times \mathsf{r}$



Chebyshev FMM : Interpolation based Matrix Vector

$$f = K\,w \approx S_x\,\bar{K}\,S_y^\top\,w$$

P2M/M2M:

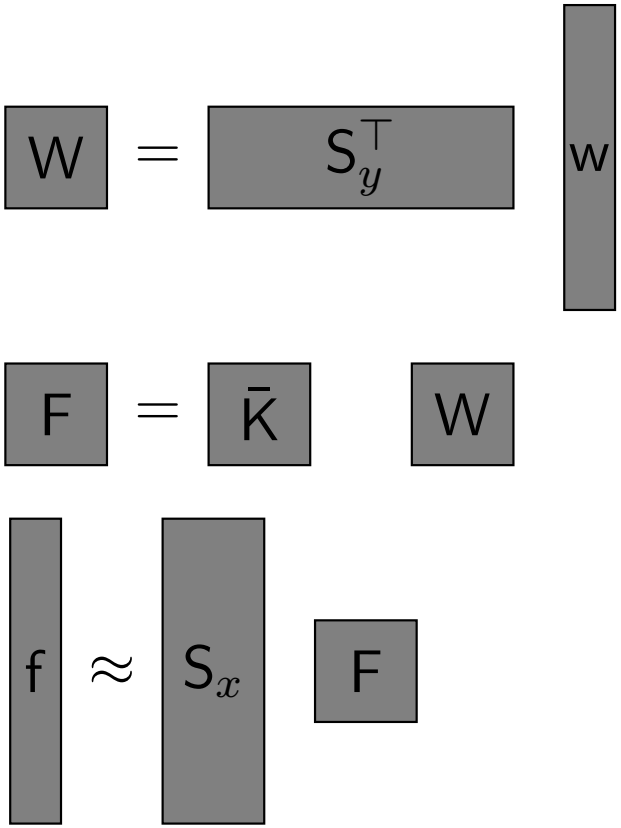
- $W = S_y^\top\,w$

M2L:

- $F = \bar{K}\,W$

L2L/L2P:

- $f \approx S_x\,F$



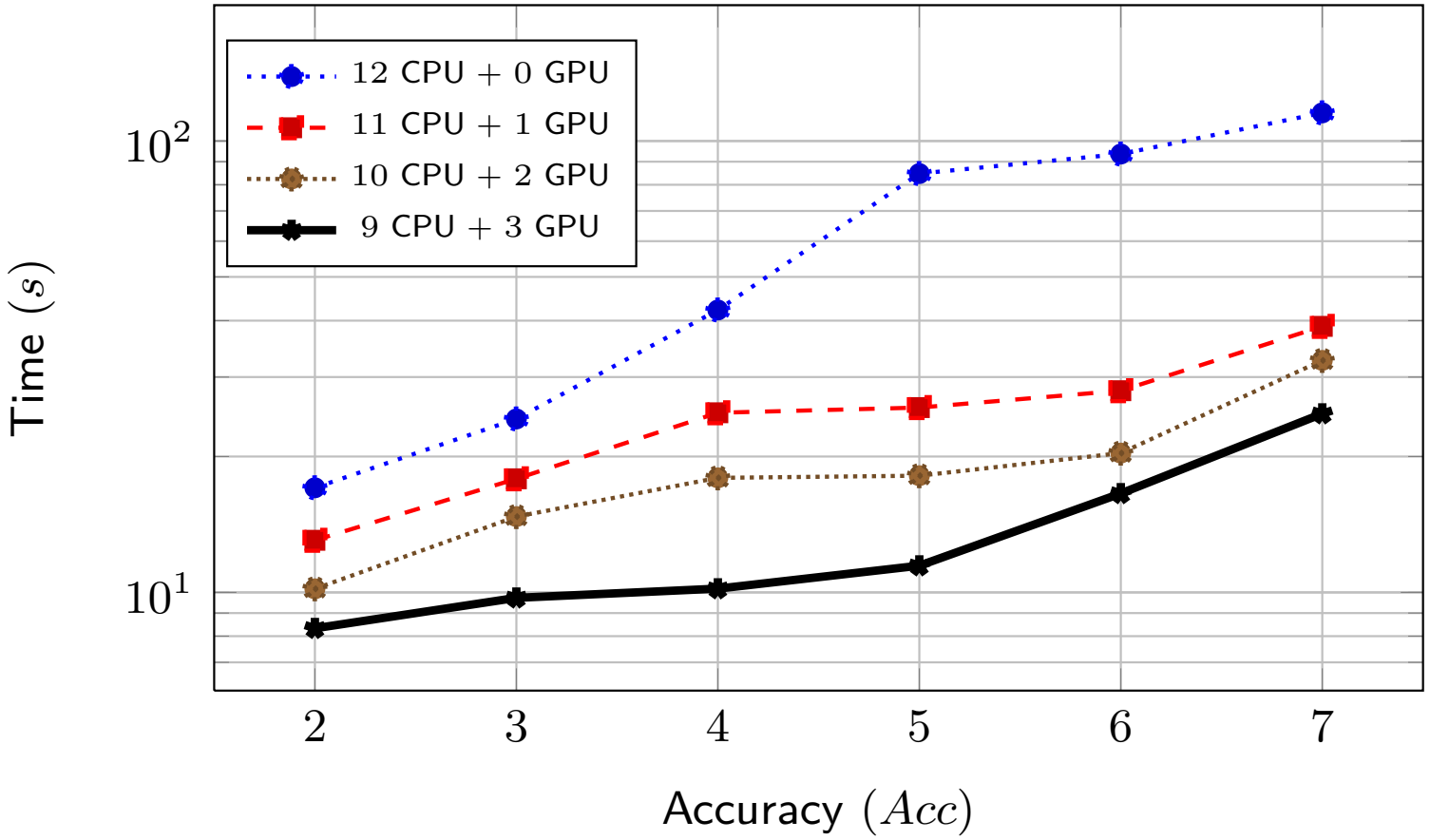
Task : Introduction

A task is composed by:

- A block of instructions
- Some parameters
- An environment
- Tasks is native in many "libraries" (OpenMP 3.1, StarPU)

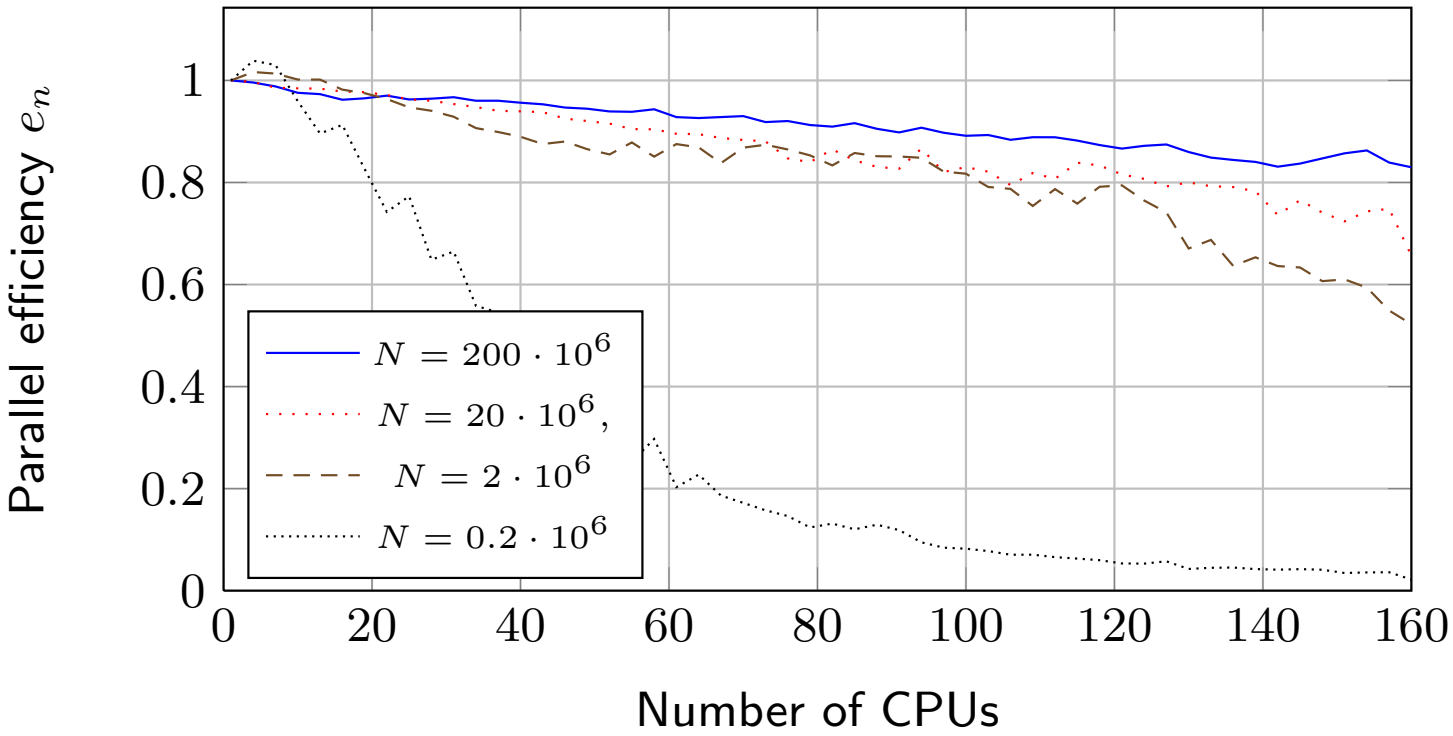
Results M2070

30.10^6 particles inside a box, granularity $n_g = 2000$, for 0 to 3 GPU and accuracy from 2 to 7.



160 Cores efficiency

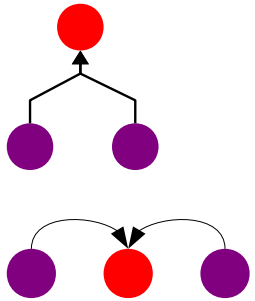
$\{N = 0.2 \cdot 10^6, n_g = 25, h = 5\}$, $\{N = 2 \cdot 10^6, n_g = 200, h = 6\}$
$\{N = 20 \cdot 10^6, n_g = 1000, h = 7\}$, $\{N = 200 \cdot 10^6, n_g = 3000, h = 8\}$



StarPU : Creating tasks

Examples:

```
insert_task(m2m, READ, children, WRITE, cell);
```



```
insert_task(m2l, READ, interactions, WRITE, cell);
```

Order of insertion matters:

- It leads to the DAG
- Different orders can create different DAG